MODULE *DistributedLock*

EXTENDS *Naturals*, *FiniteSets*, *Sequences*, *TLC*

The set of clients
CONSTANT *Clients*

Client states
CONSTANTS *Active*, *Inactive*

Message types
CONSTANT *LockRequest*, *LockResponse*, *TryLockRequest*, *TryLockResponse*, *UnlockRequest*, *UnlockResponse*

An empty constant
CONSTANT *Nil*

The current lock holder
VARIABLE *lock*

The lock queue
VARIABLE *queue*

The current lock *ID*
VARIABLE *id*

$serverVars \triangleq \langle lock, id, queue \rangle$

Client states
VARIABLE *clients*

$clientVars \triangleq \langle clients \rangle$

Client messages
VARIABLE *messages*

Variable
VARIABLE *messageCount*

$messageVars \triangleq \langle messages, messageCount \rangle$

The invariant checks that:
* No client can hold more than one lock at a time
* No two clients hold a lock with the same *ID*
* The lock is held by an active session

Note that more than one client may believe itself to hold the lock at the same time, *e.g.* if a client's session has expired but the client hasn't been notified, but lock *IDs* must be unique and monotonically increasing.

$TypeInvariant \triangleq$

$\land \forall\, c \in \text{DOMAIN } clients : Cardinality(clients[c].locks) \in 0 \,..\, 1$
$\land \forall\, c1,\ c2 \in \text{DOMAIN } clients : c1 \neq c2 \Rightarrow Cardinality(clients[c1].locks \cap clients[c2].locks) = 0$
$\land\ \lor\ \land\ lock \neq Nil$
$\quad\land\ clients[lock.client].state = Active$
$\quad\lor\ lock = Nil$

Returns a sequence with the head removed
$Pop(q) \triangleq SubSeq(q,\ 2,\ Len(q))$

Sends a message on the given client's channel
$Send(m,\ c) \triangleq$
$\quad\land\ messages' = [messages \text{ EXCEPT } ![c] = Append(messages[c],\ m)]$
$\quad\land\ messageCount' = messageCount + 1$

Removes a message from the given client's channel
$Accept(m,\ c) \triangleq$
$\quad\land\ messages' = [messages \text{ EXCEPT } ![c] = Pop(messages[c])]$
$\quad\land\ messageCount' = messageCount + 1$

Removes the last message and appends a message to the given client's channel
$Reply(m,\ c) \triangleq$
$\quad\land\ messages' = [messages \text{ EXCEPT } ![c] = Append(Pop(messages[c]),\ m)]$
$\quad\land\ messageCount' = messageCount + 1$

Handles a lock request. If the lock is not currently held by another process, the lock is granted to the client. If the lock is held by a process, the request is added to a queue.
$HandleLockRequest(m,\ c) \triangleq$
$\quad\lor\ \land\ clients[c].state = Inactive$
$\qquad\land\ Accept(m,\ c)$
$\qquad\land\ \text{UNCHANGED } \langle clientVars,\ serverVars\rangle$
$\quad\lor\ \land\ clients[c].state = Active$
$\qquad\land\ lock = Nil$
$\qquad\land\ lock' = m\ @@\ (\text{“client”} :> c)$
$\qquad\land\ id' = id + 1$
$\qquad\land\ Reply([type \mapsto LockResponse,\ acquired \mapsto \text{TRUE},\ id \mapsto id'],\ c)$
$\qquad\land\ \text{UNCHANGED } \langle queue,\ clientVars\rangle$
$\quad\lor\ \land\ clients[c].state = Active$
$\qquad\land\ lock \neq Nil$
$\qquad\land\ queue' = Append(queue,\ m\ @@\ (\text{“client”} :> c))$
$\qquad\land\ Accept(m,\ c)$
$\qquad\land\ \text{UNCHANGED } \langle lock,\ id,\ clientVars\rangle$

Handles a *tryLock* request. If the lock is not currently held by another process, the lock is granted to the client. Otherwise, the request is rejected.

$HandleTryLockRequest(m,\ c) \triangleq$
$\lor \land clients[c].state = Inactive$
$\land Accept(m,\ c)$
$\land$ UNCHANGED $\langle clientVars,\ serverVars\rangle$
$\lor \land clients[c].state = Active$
$\land lock = Nil$
$\land lock' = m \mathbin{@@} (\text{“client”} :> c)$
$\land id' = id + 1$
$\land Reply([type \mapsto LockResponse,\ acquired \mapsto \text{TRUE},\ id \mapsto id'],\ c)$
$\land$ UNCHANGED $\langle queue,\ clientVars\rangle$
$\lor \land clients[c].state = Active$
$\land lock \neq Nil$
$\land Reply([type \mapsto LockResponse,\ acquired \mapsto \text{FALSE}],\ c)$
$\land$ UNCHANGED $\langle clientVars,\ serverVars\rangle$

Handles an unlock request. If the lock is currently held by the given client, it will be unlocked. If any client's requests are pending in the queue, the next lock request will be removed from the queue and the lock will be granted to the requesting client.

$HandleUnlockRequest(m,\ c) \triangleq$
$\lor \land clients[c].state = Inactive$
$\land Accept(m,\ c)$
$\land$ UNCHANGED $\langle clientVars,\ serverVars\rangle$
$\lor \land clients[c].state = Active$
$\land lock = Nil$
$\land Accept(m,\ c)$
$\land$ UNCHANGED $\langle clientVars,\ serverVars\rangle$
$\lor \land clients[c].state = Active$
$\land lock \neq Nil$
$\land lock.client = c$
$\land lock.id = m.id$
$\land \lor \land Len(queue) > 0$
$\land$ LET $next \triangleq Head(queue)$
IN
$\land lock' = next$
$\land id' = id + 1$
$\land queue' = Pop(queue)$
$\land Reply([type \mapsto LockResponse,\ acquired \mapsto \text{TRUE},\ id \mapsto id'],\ c)$
$\lor \land Len(queue) = 0$
$\land lock' = Nil$
$\land Accept(m,\ c)$
$\land$ UNCHANGED $\langle queue,\ id\rangle$
$\land$ UNCHANGED $\langle clientVars\rangle$

Returns whether the client associated with the given message is active

$IsActive(m) \triangleq clients'[m.client].state = Active$

Expires a client's session. If the client currently holds the lock, the lock will be released and the lock will be granted to another client if possible. Additionally, pending lock requests from the client will be removed from the queue.

$$
\begin{array}{l}
ExpireSession(c) \triangleq \\
\quad \land clients[c].state = Active \\
\quad \land clients' = [clients \text{ EXCEPT } ![c].state = Inactive] \\
\quad \land \text{IF } lock \neq Nil \land lock.client = c \text{ THEN} \\
\qquad \text{LET } q \triangleq SelectSeq(queue, IsActive) \\
\qquad \text{IN} \\
\qquad\quad \lor \land Len(q) > 0 \\
\qquad\qquad \land lock' = Head(q) \\
\qquad\qquad \land id' = id + 1 \\
\qquad\qquad \land queue' = Pop(q) \\
\qquad\qquad \land Send([type \mapsto LockResponse,\ acquired \mapsto \text{TRUE},\ id \mapsto id'],\ lock'.client) \\
\qquad\quad \lor \land Len(queue) = 0 \\
\qquad\qquad \land lock' = Nil \\
\qquad\qquad \land queue' = \langle\rangle \\
\qquad\qquad \land \text{UNCHANGED } \langle id,\ messageVars\rangle \\
\quad \text{ELSE} \\
\qquad \land queue' = SelectSeq(queue, IsActive) \\
\qquad \land \text{UNCHANGED } \langle lock,\ id,\ messageVars\rangle
\end{array}
$$

Sends a lock request to the cluster with a unique *ID* for the client.

$$
\begin{array}{l}
Lock(c) \triangleq \\
\quad \land clients[c].state = Active \\
\quad \land Send([type \mapsto LockRequest,\ id \mapsto clients[c].next],\ c) \\
\quad \land clients' = [clients \text{ EXCEPT } ![c].next = clients[c].next + 1] \\
\quad \land \text{UNCHANGED } \langle serverVars\rangle
\end{array}
$$

Sends a try lock request to the cluster with a unique *ID* for the client.

$$
\begin{array}{l}
TryLock(c) \triangleq \\
\quad \land \quad clients[c].state = Active \\
\quad \land \quad Send([type \mapsto TryLockRequest,\ id \mapsto clients[c].next],\ c) \\
\quad \land \quad clients' = [clients \text{ EXCEPT } ![c].next = clients[c].next + 1] \\
\quad \land \quad \text{UNCHANGED } \langle serverVars\rangle
\end{array}
$$

Sends an unlock request to the cluster if the client is active and current holds a lock.

$$
\begin{array}{l}
Unlock(c) \triangleq \\
\quad \land clients[c].state = Active \\
\quad \land Cardinality(clients[c].locks) > 0 \\
\quad \land Send([type \mapsto UnlockRequest,\ id \mapsto \text{CHOOSE } l \in clients[c].locks : \text{TRUE}],\ c) \\
\quad \land clients' = [clients \text{ EXCEPT } ![c].locks = clients[c].locks \setminus \{\text{CHOOSE } l \in clients[c].locks : \text{TRUE}\}]
\end{array}
$$

$\land$ UNCHANGED $\langle serverVars \rangle$

Handles a lock response from the cluster. If the client's session is expired, the response is ignored. If the lock was acquired successfully, it's added to the client's lock set.

$HandleLockResponse(m,\ c) \triangleq$
$\land \lor \land clients[c].state = Inactive$
$\land$ UNCHANGED $\langle clientVars,\ serverVars \rangle$
$\lor \land clients[c].state = Active$
$\land m.acquired$
$\land clients' = [clients$ EXCEPT $![c].locks = clients[c].locks \cup \{m.id\}]$
$\land$ UNCHANGED $\langle serverVars \rangle$
$\lor \land clients[c].state = Active$
$\land \neg m.acquired$
$\land$ UNCHANGED $\langle clientVars,\ serverVars \rangle$
$\land Accept(m,\ c)$

Receives a message from/to the given client from the head of the client's message queue.

$Receive(c) \triangleq$
$\land\ Len(messages[c]) > 0$
$\land$ LET $message \triangleq Head(messages[c])$
IN
$\lor \land message.type = LockRequest$
$\land HandleLockRequest(message,\ c)$
$\lor \land message.type = LockResponse$
$\land HandleLockResponse(message,\ c)$
$\lor \land message.type = TryLockRequest$
$\land HandleTryLockRequest(message,\ c)$
$\lor \land message.type = UnlockRequest$
$\land HandleUnlockRequest(message,\ c)$

Initial state predicate

$Init \triangleq$
$\land messages = [c \in Clients \mapsto \langle\rangle]$
$\land messageCount = 0$
$\land lock = Nil$
$\land queue = \langle\rangle$
$\land id = 0$
$\land clients = [c \in Clients \mapsto [state \mapsto Active,\ locks \mapsto \{\},\ next \mapsto 1]]$

Next state predicate

$Next \triangleq$
$\lor \exists\, c \in$ DOMAIN $clients : Receive(c)$

$\lor \exists\, c \in \text{DOMAIN } clients : Lock(c)$
$\lor \exists\, c \in \text{DOMAIN } clients : TryLock(c)$
$\lor \exists\, c \in \text{DOMAIN } clients : Unlock(c)$
$\lor \exists\, c \in \text{DOMAIN } clients : ExpireSession(c)$

The specification includes the initial state predicate and the next state

$Spec \triangleq Init \land \Box[Next]_{\langle serverVars,\, clientVars,\, messageVars \rangle}$

* Modification History
* Last modified Sat *Jan* 27 02:37:15 *PST* 2018 by *jordanhalterman*
* Created *Fri Jan* 26 13:12:01 *PST* 2018 by *jordanhalterman*